ON THE CENTERPOINT PROBLEM IN COMPUTATIONAL GEOMETRY

by

Ravi Mohan H.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

February 1994

ON THE CENTERPOINT PROBLEM IN COMPUTATIONAL GEOMETR

A thesis submitted
in partial fulfilment of the requirements
for the degree of

Master of Technology

by

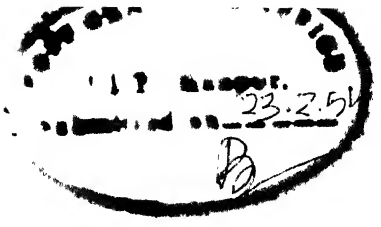
Ravi Mohan H.

to the

Department of Computer Science and Engineering

Indian Institute of Technology, Kanpur

February, 1994



CERTIFICATE

It is certified that the work contained in the thesis entitled *On the Centerpoint Problem in Computational Geometry*, by Ravi Mohan H. has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

February, 1994

Dr. Asish Mukhopadhyay,
Associate Professor,
Department of Computer Science
and Engineering,
IIT Kanpur

ABSTRACT

Efficient algorithms for computing a centerpoint have important applications in Computational Geometry. We explore these applications in this thesis. In particular, we discuss a randomised algorithm which uses the notion of a centerpoint, for finding a small graph separator of an overlap graph of a neighborhood system. Next we present an optimal parallel algorithm for computing a centerpoint of a finite planar point set in the exclusive read, exclusive write parallel random access machine model. The algorithm uses $\frac{n}{\log^3 n \log^* n}$ processors and takes $O(\log^3 n \log^* n)$ time. Finally we extend the notion of a centerpoint to the weighted case. We show that a weighted centerpoint always exists for any configuration of points with arbitrary weight assignments. We then present a linear time algorithm for computing a weighted centerpoint of a configuration of points that form the vertices of a convex polygon. In addition, we give an $O(n \log^3 n)$ algorithm for finding a weighted centerpoint of a general configuration of points.

ACKNOWLEDGEMENTS

I thank Dr.Asish Mukhopadhyay, my thesis supervisor for his guidance, inspiration and encouragement during the course of this thesis work. He introduced me to the fascinating field of Computational Geometry and motivated me to work on the centerpoint problem. In spite of his busy schedule, he always found time to spend with me to help me out of my problems, clear my doubts and set my work in the right direction.

Thanks to friends and classmates for making my stay at IIT Kanpur an enjoyable one. My association with the members and families of Kannada Sangha is indeed memorable one. My special thanks to them all for the nice time I had with them and for making me feel at home.

Contents

List of Figures

Chapter 1

Introduction

1.1 The centerpoint problem

The center of every centrally symmetric figure has the property that every chord through it bisects the area and the bounding curve of the figure; the chord itself is bisected at the center. Clearly there are planar figures having no center of symmetry - a point which bisects every chord passing through it.

In daily life, we often use phrases like "the very center of a city", even when a city has no center in any exact sense (as above). We all have an intuitive understanding as to what these phrases mean. The notion of the center of a finite set of points captures this intuition in a quantitative way for point sets. Similar notions of center exist for bounded planar curves, bounded planar figures *et cetera* [YB61].

Let $\mathcal{S}$ be a set of n points in d-dimensional Euclidean space.

Definition 1.1 *A point x in d-space, not necessarily in $\mathcal{S}$, is a centerpoint of $\mathcal{S}$ if every closed half-space including x contains at least $\lceil \frac{n}{(d+1)} \rceil$ points of $\mathcal{S}$.*

Definition 1.2 *The center of $\mathcal{S}$ is the collection of all centerpoints.*

Using the above definition, we can deduce an alternate characterisation of the center as the intersection of all closed half-spaces, each containing more than $\lfloor \frac{dn}{(d+1)} \rfloor$ points of $\mathcal{S}$.

The center is always nonempty for any configuration of points. This can be proved using the above alternate characterisation and Helly's theorem. We have the following theorem :

Theorem 1.1 *Every finite set of points in d-dimensional Euclidean space admits a centerpoint.*

The computation of a centerpoint of a finite set of points in two or higher dimensions is of fundamental importance in geometric algorithms that require a balanced partitioning of the input point set. We give a brief survey of the previous work done regarding the computation of a centerpoint of a finite point set in two and three dimensions :

1. Using the powerful technique of *Parametric Searching* [Meg83a], Cole *et al* have given an $O(n \log^5 n)$ algorithm for computing a centerpoint [CSY87] in two dimensions. Subsequently Cole improved it to $O(n \log^3 n)$ by using a refinement of the *Parametric Searching* technique [Col87]. The algorithm has been extended to three dimensions by Naor and Sharir [NS90] with a time complexity of $O(n^2 \log^6 n)$.

2. Matoušek proposed an $O(n \log^4 n)$ algorithm for computing the whole center of a set of n points in the plane [Mat92]. In addition, he gave a linear time algorithm for finding an approximate centerpoint, as close to the exact one as we want in any fixed dimension [Mat91].

3. Very recently, Jadhav and Mukhopadhyay gave a linear time prune-and-search algorithm for computing a centerpoint in two dimensions [JM93]. Their algorithm uses another important notion in Computational Geometry - that of a ham-sandwich cut.

1.2 The ham-sandwich cut problem

We first define the notion of a bisecting hyperplane. Let $\mathcal{P}$ be a set of n points in E^d.

Definition 1.3 *A hyperplane h is said to bisect $\mathcal{P}$ if neither of the two open half-spaces defined by h contain more than $\frac{n}{2}$ points of $\mathcal{P}$.*

Definition 1.4 *A ham-sandwich cut is a hyperplane that simultaneously bisects d point sets in E^d.*

The ham-sandwich cut theorem guarantees the existence of such a cut.

Theorem 1.2 *Let $P_1, P_2, \ldots, P_d$ be d finite sets of points in E^d. There exists a hyperplane h that simultaneously bisects $P_1, P_2, \ldots, P_d$.*

In two dimensions, a ham-sandwich cut is a line that simultaneously bisects both the given point sets. The generalised ham-sandwich cut problem asks for a line that splits the

two sets in specified ratios. When the point sets are linearly separated, it is always possible to find such a line.

We can use the standard transformation [Ede87] which maps points to lines and then look at the problem in the dual plane (See appendix A). If the given two point sets are linearly separated, then in the dual plane, we have two sets of lines: one having lines with positive slopes and other having lines with negative slopes. The ham-sandwich cut in the primal plane corresponds to the point where the *median* levels intersect in the dual plane. Any level of the first set is a monotonically increasing function and that of the second set is a monotonically decreasing function and hence always intersect. The generalised ham-sandwich cut problem requires one to find the point where the p^{th} level of the first set and the q^{th} level of the second set meet, given p and q.

Megiddo gave a linear time algorithm for computing a (generalised) ham-sandwich cut in the linearly separated case [Meg85]. Subsequently Lo and Steiger improved it to the general case [LS90].

1.3 Our thesis

In chapter 2, we explore the applications of computing a centerpoint, particularly in finding graph separators. Shang-Hua Teng [Tng91] has described an algorithm for finding a graph separator of a geometric graph, known as *overlap graph*, which uses the notion of a centerpoint. We discuss his algorithm and then outline the applications of computing graph separators in Numerical Analysis - for solving sparse linear systems of equations.

In chapter 3, we present an optimal parallel algorithm for finding a centerpoint of a finite point set in two dimensions, in the exclusive read, exclusive write parallel random access machine model. The algorithm takes $O(\log^3 n \log^* n)$ time using $\frac{n}{\log^3 n \log^* n}$ processors. It uses, as a subroutine, an algorithm for finding a generalised ham-sandwich cut(separable case). We present an optimal parallel algorithm for this as well, taking $O(\log^2 N \log^* N)$ time using $\frac{N}{\log^2 N \log^* N}$ processors where N is sum of the cardinalities of the two sets involved.

In chapter 4, we generalise the notion of a centerpoint of a finite planar set of points. Each point in the set is associated with a positive real number called its weight and the notion of a weighted centerpoint is defined in an analogous manner to that of an ordinary centerpoint. We show that a weighted centerpoint always exists for any configuration of

points with arbitrary weight assignments and give a linear time algorithm for computing a weighted centerpoint when the input points are the vertices of a convex polygon. Finally, we generalise Cole *et al*'s algorithm to work within the same time bound for any weighted configuration of points.

Chapter 2

Applications of Computing a Centerpoint

2.1 Introduction

Computing small graph separators has important applications in Numerical Analysis and Computational Geometry. The most classical applications of the separator results are Nested Dissection [AG73] and Generalised Nested Dissection [LRT79]- techniques used for solving large sparse linear systems of equations. Shang-Hua Teng, in his Ph.D thesis [Tng91], proposed a new class of geometric graphs, known as overlap graphs, which includes, *inter alia*, planar graphs and meshes as special cases. This class of graphs is defined based on elementary geometric objects such as points, balls etc. He described an algorithm for computing a small separator of an overlap graph, which uses the notion of a centerpoint. This algorithm will be discussed in section 2.3. In section 2.4, we outline the applications of computing graph separators in Numerical Analysis- for solving sparse linear systems of equations.

2.2 Graph separators

A set of vertices whose removal disconnects a given graph is known as a vertex separator. Edge separators are also defined similarly.

Definition 2.1 *A subset of vertices C of a graph G with n vertices is an $f(n)$-separator that δ-splits if $|C| \leq f(n)$ and the vertices of $G - C$ can be partitioned into two sets A*

and B such that $|A|, |B| \leq \delta n$ and there is no edge between A and B, where f is a positive function and $0 < \delta < 1$.

When we say that a graph G has a small separator, we mean that there is a constant δ, $0 < \delta < 1$, and a sublinear function f such that G has an $f(n)$ separator that δ-splits. We say that a class of graphs has an $f(n)$-separator theorem if there exist constants $\delta < 1$ and $\beta > 0$ such that every graph in the class has a $\beta f(n)$-separator that δ-splits.

Divide and Conquer paradigm, when applied to graph problems, requires partitioning the given graph by removing a subset of vertices or edges. The subproblems are those associated with the connected components of the resulting graph. The cost of combining usually depends on the size of the separator used.

2.3 Application of centerpoints in finding graph separators

In this section, we introduce the notion of overlap graphs and show that the class of overlap graphs includes grids, planar graphs and k-nearest neighborhood graphs as special cases. We then discuss an algorithm for finding a small separator of an overlap graph. The results in this section are from [Tng91].

2.3.1 Some simple classes of graphs

One of the simplest classes of geometric graphs is the class of grid graphs. In two dimensions, they are also known as meshes. Another example is the class of k-nearest neighborhood graphs- which has important applications in Computational Geometry.

Let $\mathcal{P} = \{p_1, \ldots, p_n\}$ be a set of n points in $\Re^d$. Let $N_k(p_i)$ denote the set of k nearest neighbors of p_i; ties are broken arbitrarily.

Definition 2.2 *A k-nearest neighborhood graph of $\mathcal{P} = \{p_1, \ldots, p_n\}$ in $\Re^d$ is a graph with vertices $\mathcal{V} = \{1, \ldots, n\}$ and edges $\mathcal{E}$ where $\mathcal{E} = \{(i,j) \mid p_i \in N_k(p_j)$ or $p_j \in N_k(p_i)\}$.*

Yet another class of graphs in which we would be interested in this section is the class of planar graphs.

2.3.2 Neighborhood systems and overlap graphs

We now give a sequence of definitions and results - either omitting the proof fully or only giving a brief sketch of it. The details can be found in [Tng91].

Definition 2.3 *An Euclidean neighborhood of a given point $p \in \Re^d$ is a closed ball of certain radius centered at p. A neighborhood system $\Xi = \{B_1, \ldots, B_n\}$ is a finite collection of neighborhoods.*

Let $\Xi = \{B_1, \ldots, B_n\}$ be a neighborhood system. Let p_i be the center of B_i, $1 \leq i \leq n$. $\mathcal{P} = \{p_1, \ldots, p_n\}$ is called the *centers* of Ξ.

Definition 2.4 *For any integer k, Ξ is a k-neighborhood system if, for all i, $1 \leq i \leq n$, the interior of B_i contains no more than k points from $\mathcal{P}$.*

Given a ball B of radius r and a positive real number α, $\alpha.B$ denotes the ball with the same center but radius αr. If $\alpha \geq 1$, then the ball $\alpha.B$ is known as the α-dilation of B.

Definition 2.5 *The kissing number in d-dimensions, denoted by τ_d, is defined to be the maximum number of non-overlapping unit balls in $\Re^d$ that can be arranged so that they all touch a central unit ball.*

The kissing numbers in one, two and three dimensions are 2, 6 and 12 respectively.

Given a neighborhood system Ξ and a point $p \in \Re^d$, $density_\Xi(p)$ denotes the number of neighborhoods in Ξ that contain p (either in the interior or on the boundary).

Lemma 2.1 (Density Lemma) *For each k-neighborhood system $\Xi = \{B_1, \ldots, B_n\}$ in d dimensions, for each $p \in \Re^d$, $density_\Xi(p) \leq \tau_d k$.*

Proof: See [Tng91]. □

Given a neighborhood system, two graphs can be defined - an intersection graph and an α-overlap graph ($\alpha \geq 1$). Both have the same set of vertices i.e. one vertex per ball. The intersection graph has an edge between two vertices whenever the corresponding balls intersect. The α-overlap graph, $\alpha \geq 1$, has an edge between the vertices corresponding to two balls if the α-dilation of the smaller ball intersects the larger ball. Note that the intersection graph is the same as 1-overlap graph.

We now prove that the class of overlap graphs includes grids, planar graphs and k-nearest neighborhood graphs as special cases. To precisely define what the above statement means, we introduce the notion of an embedding.

An embedding of a graph $G = (V, E)$ in $\Re^d$ is a mapping $\pi : V \rightarrow \Re^d$. An edge (u, v) of G is mapped to the line segment $(\pi(u), \pi(v))$.

Definition 2.6 *A graph $G = (V, E)$ is k-embeddable in $\Re^d$ if there is an embedding π of G such that for all $(u, v) \in E$, $B_u \cap B_v \neq \phi$, where B_u is the largest ball centered at $\pi(u)$ that contains no more than k points from $\{\pi(w) : w \in V\}$ in its interior. Similarly, $G = (V, E)$ is (α, k)-embeddable in $\Re^d$ if there is an embedding π of G such that for all $(u, v) \in E$, $B_u \cap (\alpha.B_v) \neq \phi$ and $(\alpha.B_u) \cap B_v \neq \phi$.*

The above definition implies that

1. A graph G is k-embeddable if G is a subgraph of the intersection graph of some k-neighborhood system.

2. A graph G is (α, k)-embeddable if it is a subgraph of the α-overlap graph of some k-neighborhood system.

Theorem 2.1 *The class of overlap graphs includes grid graphs in d-dimensions.*

Proof: Given a grid graph, we define a 1-embedding in d-space as follows : At each vertex of the grid, we put a ball with radius $\frac{1}{2}$. The given grid graph is the intersection graph of this neighborhood system. □

Theorem 2.2 *The class of overlap graphs includes planar graphs.*

Proof: See [Tng91]. □

Theorem 2.3 *The class of overlap graphs includes k-nearest neighborhood graphs in d-dimensions.*

Proof: Let P be the given set of n points in d-space. A k-embedding is defined by letting the points to map to themselves; B_i denotes the largest ball centered at p_i whose interior contains no more than k points from P. According to the definition of k-nearest neighborhood graphs, there is an edge between points p_i and p_j only if either p_i is in B_j or

p_j is in B_i. Whenever this is the case, B_i and B_j intersect and hence the intersection graph has an edge between p_i and p_j. Thus it follows that each k-nearest neighborhood graph in d-dimensions is k-embeddable in d-space. □

Thus to find a separator for a given graph (from the above classes), we embed it in d-space and find a neighborhood system, the overlap graph of which is a "super graph" of the given graph. We then proceed to compute a separator for this overlap graph, which will be a separator for the original graph as well.

2.3.3 Sphere separators

Each sphere S in $\Re^d$ separates $int(S)$ from $ext(S)$ in the sense that any segment connecting a point in $int(S)$ with one in $ext(S)$ must intersect S and is known as a sphere separator. If P is a set of n points in $\Re^d$ and δ a constant, $0 < \delta < 1$, then S δ-splits P if both $|int(S) \cap P| \leq \delta n$ and $|ext(S) \cap P| \leq \delta n$.

Cost of a sphere separator

Just as a vertex separator has a cost (usually the number of vertices in the separator), a sphere separator is also associated with a cost - the surface area of the sphere being the most natural choice. The surface area may either be weighted or unweighted.

The unweighted surface area of a sphere in d-space with radius r is given by

$$\frac{2\pi^{d/2} r^{d-1}}{\Gamma(d/2)}$$

where Γ is the gamma function.

In the weighted case, there is a real valued non-negative function $f(x)$ such that f^k is integrable for all $k = 1, 2, 3, \ldots$. Such an f is called a density function. It may be defined on $\Re^d$ or on a unit d-sphere in $\Re^{d+1}$. We consider the case where f is defined on a unit d-sphere, say U_d, in $\Re^{d+1}$. The total volume of f is defined to be

$$total\_volume(f) = \int_{v \in U_d} (f(v))^d (dv)^d$$

A great sphere of U_d is the intersection of U_d with a hyperplane passing through the center of U_d. The weighted area of a great sphere GS of U_d is given by

$$Area_f(GS) = \int_{v \in GS} (f(v))^{d-1} (dv)^{d-1}$$

Let $avg(f)$ be the average area over all great spheres of U_d. We have the following proposition :

Proposition 2.1 *Suppose f is a density function on a unit d-sphere U_d, then $avg(f) = A_{d-1}\left((total\_volume(f))^{\frac{d-1}{d}}\right)$, where A_d stands for the surface area of U_d.*

Proof: See [Tng91]. □

Intersection number

Let Ξ be a neighborhood system in d-space. Each $(d-1)$-dimensional sphere S partitions the balls in Ξ into three sets : those which are completely in the interior of S (Ξ_I), those which are completely in the exterior of S (Ξ_E) and those which intersect S (Ξ_O). The cardinality of Ξ_O is known as the intersection number of S and is denoted by $\iota_\Xi(S)$.

Removal of the balls in Ξ_O splits Ξ into two subsets Ξ_I, Ξ_E such that no ball in Ξ_I intersects any ball in Ξ_E and *vice versa*. It follows that the removal of the vertices corresponding to the balls in Ξ_O from the intersection graph of Ξ separates the graph. So we have the lemma :

Lemma 2.2 *If S is a sphere that δ-splits the centers of a neighborhood system Ξ, then the vertices corresponding to Ξ_O defined above is an $\iota_\Xi(S)$-separator that δ-splits the intersection graph of Ξ.*

So to find a small separator for the intersection graph of a neighborhood system Ξ that δ-splits, one finds a sphere separator S which δ-splits the centers of Ξ and having a low intersection number. The vertices corresponding to the balls intersected by S give the vertex separator and the intersection number is the cardinality of the vertex separator. However these vertices need not separate the α-overlap graph of Ξ. To separate the α-overlap graph, we remove, in addition to the above vertices, the vertices corresponding to the balls which satisfy the following two conditions :

1. Their radii should be less than or equal to that of the radius of the sphere separator S.

2. Their α-dilations should intersect S.

The total number of such vertices is called the overlap number of S and is denoted by $\vartheta_{\Xi}(S)$. Note that in both the cases, the vertex separator can be found from the neighborhood system and the sphere separator in linear time.

Existence of a sphere separator with low cost

The stereographic projection is a mapping which maps $\Re^d$ plus infinity onto the unit d-sphere in $\Re^{d+1}$ with origin as the center, denoted by U_d. The pre-image of a great sphere GS of U_d is a $(d-1)$-sphere, say S, in $\Re^d$. Moreover the interior and the exterior of S are mapped to the two hemispheres of U_d, defined by GS. S δ-splits a point set P iff GS δ-splits Q, the image of P under stereographic projection. Now if the origin is chosen to be a centerpoint of Q, then the pre-image of any great sphere will $\frac{d+1}{d+2}$-split P. Based on these ideas and using proposition 2.1, Miller and Thurston derived the following continuous separator theorem when the weighted surface area is used as the cost function of a sphere.

Theorem 2.4 (Miller and Thurston) *Suppose f is a density function on $\Re^d$ and P a set of n distinct points in $\Re^d$. Then there is a sphere S which $\frac{d+1}{d+2}$-splits P such that $Area_f(S) = O\left((total\_volume(f))^{\frac{d-1}{d}}\right)$.*

Let Ξ be a d-dimensional neighborhood system with density μ. In the above theorem, P can be chosen to be the centers of Ξ. By an appropriate choice of a density function, the following theorems can be derived:

Theorem 2.5 (Intersection Graphs) *If G is the intersection graph of a d-dimensional neighborhood system with density μ, then G has an $O(\mu^{\frac{1}{d}} n^{\frac{d-1}{d}})$-separator that $\frac{d+1}{d+2}$-splits.*

Theorem 2.6 (Overlap Graphs) *If G is the α-overlap graph of a d-dimensional neighborhood system with density μ, then G has an $O(\alpha\mu^{\frac{1}{d}} n^{\frac{d-1}{d}})$-separator that $\frac{d+1}{d+2}$-splits.*

Proof: In both the cases, the density function is chosen such that

1. $total\_volume(f)$ is linearly bounded by n i.e. in the theorem on intersection graphs, $total\_volume(f) = O(\mu^{\frac{1}{d-1}} n)$ and in the case of overlap graphs, $total\_volume(f) = O(\alpha^{\frac{d}{d-1}} \mu^{\frac{1}{d-1}} n)$.

2. The weighted surface area is an upper bound on the intersection number in the case of intersection graphs and on the overlap number in the second case. □

2.3.4 A randomised algorithm for finding a sphere separator of low cost

The discussions in the previous subsections suggests the following randomised algorithm for computing a sphere separator with a small area, given a set of points P in d-space :

ALGORITHM

1. Project the points in P onto U_d (the unit d-sphere in $\Re^{d+1}$ centered at the origin o) using a mapping π, such that the origin o is a centerpoint of the projected points. Such a π is demonstrated in [MT90].

2. Randomly choose a great sphere GS of U_d.

3. Using the inverse of π, map GS back into d-space to get a $(d-1)$-sphere S.

Because o is a centerpoint of the projected points in $(d+1)$-space, the sphere S $\frac{d+1}{d+2}$-splits P. Moreover, as proved in the continuous separator theorem, with probability $\frac{1}{2}$, S has area $(total\_volume(f))^{\frac{d-1}{d}}$, where f is any density function defined on $\Re^d$.

The running time of the above algorithm depends on the time needed to compute a centerpoint in $(d+1)$-space; other steps take $O(n)$ time. However no efficient algorithms are known to compute a centerpoint in d-dimensions, $d > 2$. Using an approximate centerpoint algorithm, [Tng91] has shown how to compute a sphere separator with small area in random linear time.

2.4 Graph separators in Numerical Analysis

In this section, we discuss the applications of finding graph separators in Numerical Analysis. The classical applications are Nested Dissection [AG73] and Generalised Nested Dissection [LRT79] - techniques used for solving sparse linear systems of equations.

Suppose we wish to solve the system of linear equations $Ax = b$ by Gaussian elimination. A is an $n \times n$ symmetric, positive definite matrix, x is an $n \times 1$ vector of variables and b is an $n \times 1$ vector of constants. Usually the solution is obtained in two steps : first we factor A by means of row operations into $A = LDL^T$ where L is lower triangular and D is diagonal. Next we solve the simplified systems $Lz = b$, $Dy = z$ and $L^Tx = y$.

If A is dense, then the time required for factoring A is $O(n^3)$ and the time required for solving the simplified systems is $O(n^2)$. But if A is sparse, then we may be able to save time

and space by avoiding explicit manipulation of zeros. However, the factoring process may create nonzeros in L in positions where A had zeros. These nonzeros are called $fill - in$.

The amount of $fill - in$ can be reduced by suitably reordering the variables i.e. we transform A into $A' = PAP^T$ where P is a permutation matrix and then solve the reordered system. Let $A' = LDL^T$. Let $d(i)$ denote the number of nonzeros in column i of L. Then the number of nonzeros in L is $m = \sum_{i=1}^{n-1} d(i)$. The number of multiplications performed during the factorisation can be shown to be $\frac{1}{2}\sum_{i=1}^{n-1} d(i)(d(i)+3)$ [Ros73]. So a major issue in sparse Gaussian elimination is that of finding a good ordering of the variables which reduces the number of nonzeros in L and the multiplication count.

We can represent the given matrix A by means of an undirected graph $G = (V, E)$. The graph G contains one vertex $i \in V$ for each row of A and one edge $\{i, j\} \in E$ for each pair of nonzero, off-diagonal elements $a_{ij} = a_{ji} \neq 0$ in A. Each permutation matrix corresponds to a numbering of the vertices of G and the graph G represents the class PAP^T. See [RTL76] and [Ros73] for a discussion of the properties of this graph-theoretic model of sparse Gaussian elimination.

Finding a good ordering of the variables i.e. a numbering of the vertices for an arbitrary graph seems to be very hard. However, for some special cases, good ordering schemes are known.

1. The Nested Dissection method of A.George [AG73] allows a linear system of equations whose graph is an $n = k \times k$ square grid to be solved in $O(n^{\frac{3}{2}})$ time and $O(n \log n)$ space. His scheme uses the fact that removal of $O(k)$ vertices from a $k \times k$ grid *dissects* (i.e. separates) the grid and leaves four square grids, each roughly of size $\frac{k}{2} \times \frac{k}{2}$.

2. The above scheme has been generalised, with the same time and space bounds, to solve any system of equations, whose graph belongs to a class S, where S is closed under the subgraph relation and satisfies a $\sqrt{n}$-separator theorem [LRT79]. Since the class of planar graphs satisfies the above two conditions [LR79], it follows that any system of equations whose graph is planar can be solved in $O(n^{\frac{3}{2}})$ time and $O(n \log n)$ space. The method employed to generate the ordering in the above scheme uses an algorithm for finding a $\sqrt{n}$-separator for planar graphs.

Chapter 3

An Optimal Parallel Algorithm for Computing a Centerpoint in Two Dimensions

3.1 Introduction

The model of computation used for the parallel algorithms presented in this chapter is the exclusive-read, exclusive write (EREW) parallel random access machine, which does not allow simultaneous access by more than one processor to the same memory location , for either read or write purposes. Given a parallel algorithm, the product of the number of processors used and the time taken by the algorithm gives a measure of the number of operations performed. The parallel algorithm is said to be optimal if this is of the same order as the fastest known worst-case running time of a sequential algorithm for the problem.

In this chapter, we present an optimal parallel algorithm for finding a centerpoint of a finite point set in two dimensions. In addition, we give an optimal parallel algorithm for finding a generalised ham-sandwich cut in the plane(separable case), which is used as a subroutine in the centerpoint algorithm. Both are in the EREW model.

3.2 Preliminaries

Consider a parallel algorithm with k stages. Suppose stage i can be implemented using p_i processors in time t_i. Let $q_i = p_i * t_i$.

Let $Q = \sum_{i=1}^{k} q_i$ and $T = \sum_{i=1}^{k} t_i$. Now suppose $\lceil Q/T \rceil$ processors are available. Then stage i can be implemented to run in $max(t_i, t_i * \lceil \frac{p_i}{\lceil (Q/T) \rceil} \rceil) < \frac{p_i t_i}{\lceil (Q/T) \rceil} + 2t_i$ time. So the total time taken by the algorithm will be $\sum_{i=1}^{k} 2t_i + \sum_{i=1}^{k} (\frac{p_i t_i}{\lceil (Q/T) \rceil}) = 2T + \frac{Q}{\lceil (Q/T) \rceil} = O(T)$. Thus the entire algorithm can be executed in $O(T)$ time using $O(Q)$ operations.

We will be using the above idea, known as Brent's lemma([Br74]), in the algorithms described below. We describe the algorithms as if different number of processors are available for the different stages and determine the total number of operations performed Q and the total time taken T. It will then be possible to implement the algorithms using $\lceil (Q/T) \rceil$ processors to run in $O(T)$ time. The rescheduling of the algorithms is straight forward. See appendix B.

3.3 An optimal parallel algorithm for finding a generalised ham-sandwich cut in two dimensions

In this section, we present an optimal parallel algorithm for finding a generalised ham-sandwich cut(separable case) in the EREW model, which is a parallel version of Megiddo's algorithm [Meg85]. The problem is considered in the dual plane.

We are given two sets of lines : A=$\{y = a_i x + b_i : i$=1 to $n\}$ where each $a_i > 0$ and B=$\{y = c_i x + d_i : i$=1 to $m\}$ where each $c_i < 0$. Let $N = n + m$. Also given are two integers p and q, $1 \leq p \leq n$ and $1 \leq q \leq m$. Let $I(x^*, y^*)$ be the point where the p^{th} level of A and the q^{th} level of B meet. We are interested in finding the point $I(x^*, y^*)$.

The search for the point I is carried out as follows : we use a subroutine that determines, for any arbitrary line , the side on which I lies. In each iteration, a fixed fraction of the lines is discarded from the two sets, after identifying the side on which I lies for each of them. We do this by testing two 'special lines' in the above sense.

Given any line L:$y = \alpha x + \theta$, we show how to determine the side on which I lies in $O(\log N \log^* N)$ time using $\frac{N}{\log N \log^* N}$ processors. There are four cases : $\alpha > 0$, $\alpha < 0$, $\alpha = 0$ and $\alpha = \infty$ (i.e. L is vertical). We consider the case when $\alpha > 0$; other cases are similar.

1. Find the intersections of the m lines in B with L. This can be done in $O(\log N \log^* N)$ time using $\frac{N}{\log N \log^* N}$ processors.

2. From among the x-coordinates of these intersection points, choose the q^{th} largest one x'. This can be done in $O(\log N \log^* N)$ time using $\frac{N}{\log N \log^* N}$ processors using the selection algorithm of Cole [Col88].

3. Let $y'=\alpha x' + \theta$. Find the intersections of the lines in A with the line $x=x'$ and from among the y-coordinates of these intersection points, choose the p^{th} largest one y_p. This can be done in $O(\log N \log^* N)$ time using $\frac{N}{\log N \log^* N}$ processors.

   If $y_p=y'$, we have found I.

   If $y_p > y'$, I lies above L.

   If $y_p < y'$, I lies below L.

So the question of determining on which side of a given line I lies can be answered in $O(\log N \log^* N)$ time using $\frac{N}{\log N \log^* N}$ processors where N is the sum of the cardinalities of the two sets.

The parallel algorithm for finding the cut has $O(\log N)$ stages. In each stage, a fixed fraction $1-\beta$ ($\beta > 0$) of the lines are discarded after determining the side on which I lies for each of them. For the method described below, β turns out to be $\frac{7}{8}$.

The pruning is carried out as follows :

1. Find the median slope s of the lines (from both the sets) and then form two sets : one having lines with slope larger than s and the other having lines with slope smaller than s. Pair lines- one from each set and compute their point of intersection.

2. From among the x-coordinates of the intersection points, find the median x-coordinate x_m.

3. Test on which side of the line $x = x_m$, $I(x^*, y^*)$ lies. Say $x_m > x^*$. Let S be the set of intersection points each of whose x-coordinate is greater or equal to x_m.

4. Let S' be the set of lines of slope s through each point in S plus the lines of slope s from both the sets. Find a line of slope s which divides this S' into two equal parts and test on which side of this line I lies. Thus we identify an open quadrant Q in which I lies.

5. Each pair associated with the closed quadrant, opposite to Q has a line that does not pass through Q i.e. we know the side on which I lies for this line. In addition, there

may be some lines of slope s that avoid Q. Altogether there are at least $\frac{N}{8}$ lines which avoid Q and they can be removed from further consideration. The values of p and q are properly updated.

Each step here can be done in $O(\log X \log^* X)$ time using $\frac{X}{\log X \log^* X}$ processors where $X = N$ for the first stage, $X = \beta N$ for the second stage, $X = \beta^2 N$ for the third stage and so on. See appendix B for a detailed analysis of the algorithm. The total work done $= N + \beta N + \beta^2 N + \ldots = O(N)$. The total time taken $= \log N \log^* N + \log(\beta N)\log^*(\beta N) + \ldots = O(\log^2 N \log^* N)$. So the whole algorithm can be implemented to run in $O(\log^2 N \log^* N)$ time using $\frac{N}{\log^2 N \log^* N}$ processors.

3.4 An optimal parallel algorithm for computing a centerpoint in two dimensions

In this section, we present an optimal parallel algorithm for computing a centerpoint in the EREW model which is a parallel version of the algorithm presented in [JM93].

Let P be the given set of n points. We briefly describe the algorithm given in [JM93] for finding a centerpoint. The notation S_H denotes the set of points contained in the half-plane H and S_{GH} denotes the set of points common to the half-planes G and H. The algorithm finds four closed half-planes L, U, D and R, each containing less than $\lceil \frac{n}{3} \rceil$ points of P and situated so that the intersection of their complements is a bounded open quadrilateral or a triangle. It is also ensured, while finding these half-planes, that each of the sets S_{LU}, S_{LD}, S_{RU} and S_{RD} contain at least $\lfloor \frac{n}{12} \rfloor - 1$ points of P. Four points are then chosen, one from each of the above sets and if they form a convex quadrilateral, they are replaced by the point of the intersection of the diagonals of the quadrilateral and if they form a non-convex quadrilateral, the interior point is retained and the remaining three are discarded. As any centerpoint of the new set is also a centerpoint of the original one, the algorithm proceeds to work on the new set. Thus in each stage, a fixed fraction of the input points are discarded and the work done per stage is linear, hence the whole algorithm runs in linear time.

The four planes, mentioned above, are found as follows : from among the x-coordinates of all the points, the $(\lceil \frac{n}{3} \rceil - 1)^{th}$ one is found and the closed half-plane to the left of the perpendicular line through the point with this x-coordinate is chosen as L. Then the ham-sandwich cut algorithm is used to partition S_L in the ratio 1:3 and the set $P - S_L$ in the ratio

3:5 so that one of the closed half-planes determined by the partitioning line contains exactly $\lceil \frac{n}{3} \rceil - 1$ points of P. This closed half-plane is chosen as U. It is ensured, while finding the cut, that S_{LU} contains $\lfloor \frac{n}{12} \rfloor - 1$ points of P. The closed half-plane D is determined similarly. To find R, the sets S_U and $P - S_U$ are partitioned in a similar manner. By construction, the sets S_{LU}, S_{LD} and S_{RU} contain $\lfloor \frac{n}{12} \rfloor - 1$ points each and as shown in [JM93], the set S_{RD} contains at least $\lfloor \frac{n}{12} \rfloor - 1$ points.

The parallel algorithm has $O(\log n)$ stages. In each stage, a fraction $1 - \alpha$ of the points will be pruned as described above.

A typical stage is implemented as follows:

1. The four planes L, U, D and R are found using the selection algorithm of Cole [Col88] and the ham-sandwich cut algorithm described in section 3.3.

2. The four sets S_{LU}, S_{LD}, S_{RU} and S_{RD} are found and then the points are pruned as described before. See appendix C for more details.

Each stage can be implemented to run in $O(\log^2 X \log^* X)$ time, doing $O(X)$ work where $X = n$ for the first stage, $X = \alpha n$ for the second stage and so on. So the total work done is $O(n)$ and the total time taken is $O(\log^3 n \log^* n)$. So the whole algorithm can be implemented to run in $O(\log^3 n \log^* n)$ time using $\frac{n}{\log^3 n \log^* n}$ processors.

Chapter 4

The Notion of a Weighted Centerpoint

4.1 Introduction

There is another way of looking at the notion of a centerpoint of a planar set of points- as a generalisation of the concept of a median of a set of points on the real line. The concept of a median of a set of points on the real line has been extended to the 'weighted median' notion, where each point is associated with a positive weight and a weighted median is defined as a point which gives a balanced partition of the total weight of all the points. A linear time algorithm for finding a weighted median is given in [Rei78].

In a similar spirit, we define the notion of a weighted centerpoint of a finite planar set of points. We show that a weighted centerpoint always exists for any configuration of points with arbitrary weight assignments. We also propose a linear time algorithm for finding a weighted centerpoint of a configuration of points that form the vertices of a convex polygon (The input should be in the form of a list of vertices as they lie on the boundary of the convex polygon). Finally we give an $O(n \log^3 n)$ algorithm for finding a weighted centerpoint of any weighted configuration of points, generalising Cole *et al*'s algorithm [CSY87, Col87].

This chapter is organised as follows: In section 4.2, we give a formal definition of the notion of a weighted centerpoint and prove that such a point exists for any finite planar configuration of points. In section 4.3, we present a linear time algorithm for finding a weighted centerpoint of a configuration of points that form the vertices of a convex poly-

gon. Section 4.4 discusses an algorithm for finding a weighted centerpoint of a general configuration of points.

4.2 Definition and proof of the existence of a weighted centerpoint

We are given a set $\mathcal{P}$ of n points in the plane. Each point p_i is associated with a positive number w_i, denoting its weight. Let the total weight of all the points be W.

Definition 4.1 *A point x in the plane, not necessarily in $\mathcal{P}$, is a weighted centerpoint if for any closed half-plane containing x the sum of the weights of the points included in it is at least $\lceil \frac{W}{3} \rceil$.*

Definition 4.2 *The weighted center is the collection of all weighted centerpoints.*

We now show that a weighted centerpoint always exists for any configuration of points with arbitrary weight assignments. The proof is similar to the one given in [YB61] for the case of an ordinary centerpoint.

Let $\mathcal{R}$ be the intersection of all closed half-planes, the sum of the weights of the points included in each of which is strictly greater than $W - \lceil \frac{W}{3} \rceil (= \lfloor \frac{2W}{3} \rfloor)$.

We claim that any point belonging to $\mathcal{R}$ is a weighted centerpoint. Otherwise, if x is a point in $\mathcal{R}$ which is not a weighted centerpoint then there exists a closed half-plane whose bounding line l contains x and the sum of the weights of the points included in it is less than $\lceil \frac{W}{3} \rceil$. Consider a line m, lying in the complementary open half-plane, parallel and sufficiently close to l so that none of the points of $\mathcal{P}$ lie between l and m. The sum of the weights of the points included in the closed half-plane defined by m, which excludes the line l, is greater than $W - \lceil \frac{W}{3} \rceil$ and hence x, being a point in $\mathcal{R}$, should be in this closed half-plane - a contradiction.

It remains to show that $\mathcal{R}$ is always nonempty. We assume that the whole point set is covered by a disk and consider the infinite number of bounded convex regions obtained by intersecting each of the half-planes above with this disk.

We show that any three of these have a common point and hence by Helly's theorem it would follow that $\mathcal{R}$ is nonempty.

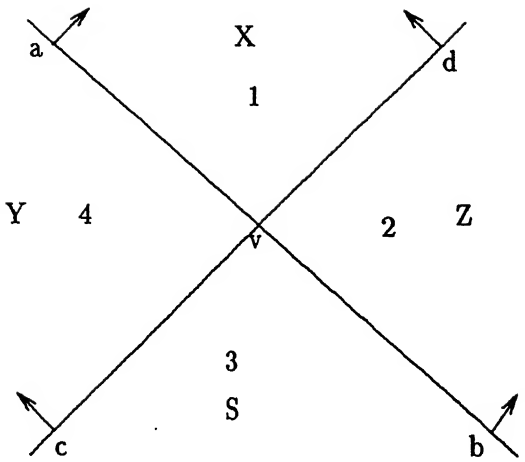


Figure 4.1: Distribution of the total weight of the points in the four quadrants

Let two of the half-planes be situated as in figure 4.1 . Quadrant 1 is closed, whereas quadrant 3 is open; X and S are respectively the sum of the weights of the points included in these quadrants. Quadrant 4 is sort of half-closed in the sense that it includes all the points on the open half-line vc. Similarly, quadrant 2 includes all the points on the open half-line vb. Y and Z are respectively the sum of the weights of the points in these quadrants.

Since $X + Y > W - \lceil \frac{W}{3} \rceil$ and $X + Z > W - \lceil \frac{W}{3} \rceil$, it follows that X is at least $W - (2\lceil \frac{W}{3} \rceil) + 2$. This means that if all the points in quadrant 1 are excluded, then the sum of the weights of the remaining points is less than or equal to $2\lceil \frac{W}{3} \rceil - 2$, which is never greater than $W - \lceil \frac{W}{3} \rceil$. So the third closed half-plane will have to include at least one point from quadrant 1 which means that any three closed half-planes have a common point.

So we have the theorem :

Theorem 4.1 *The weighted center is always nonempty for any planar configuration of points.*

4.3 An algorithm for finding a weighted centerpoint of a convex polygon

In this section, we present an algorithm for computing in linear time a weighted centerpoint of a set of points that the form the vertices of a convex polygon. The input is a list of vertices in the order in which they occur on the boundary of the convex polygon. It turns out that the weighted center itself can be computed in $O(n \log n)$ time.

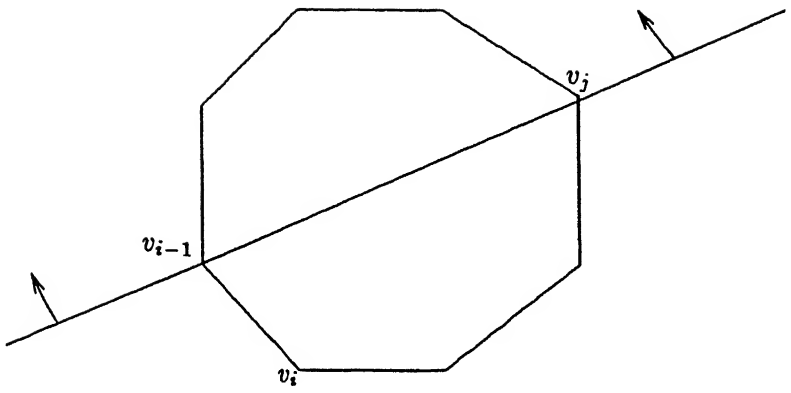


Figure 4.2: Illustrating the definition of *CHAIN*

Let $\mathcal{P}$ be a convex polygon with n vertices. The vertices are labeled from v_1 to v_n in the counterclockwise order (Vertex v_{n+1} is the same as vertex v_1). Vertex v_i has weight w_i. Let $W = \sum w_i$.

If any vertex has weight greater than $\lfloor \frac{2W}{3} \rfloor$, then that vertex is a weighted centerpoint (in fact, it is the only weighted centerpoint). This can be checked in linear time. So we assume that every $w_i \leq \lfloor \frac{2W}{3} \rfloor$.

Let $CHAIN(v_i) = v_i v_{i+1} \ldots v_j, i \leq j$ represent a counterclockwise chain of vertices, beginning from v_i such that the sum of the weights of the vertices included in it just $\geq \lceil \frac{W}{3} \rceil$. Let $CP(v_i)$ denote the closed half-plane defined by the line through the vertices v_{i-1} and v_j which does not include the vertices $v_i, v_{i+1}, \ldots, v_{j-2}$ and v_{j-1}. See figure 4.2. Such a half-plane $CP(v_i)$ is well defined for each i.

Let $\mathcal{R}$ denote the intersection of all the $CP(v_i)$'s.

Lemma 4.1 *Any point not belonging to $\mathcal{R}$ is not a weighted centerpoint.*

Proof : Let x be any such point. Since x does not belong to $\mathcal{R}$, there is some $CP(v_i)$ which does not contain x. Now consider the closed half-plane, defined by a line through x parallel to the bounding line of $CP(v_i)$, which does not include $CP(v_i)$. The sum of the weights of the points included in this closed half-plane is less than $\lceil \frac{W}{3} \rceil$. So x is not a weighted centerpoint. □

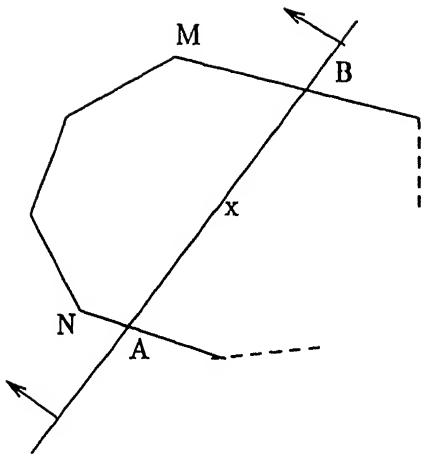


Figure 4.3: Diagram for the proof of *Lemma 4.2*

Lemma 4.2 *Any point belonging to $\mathcal{R}$ is a weighted centerpoint.*

Proof : Let x be any such point. Consider an arbitrary line through x. Since the given polygon is convex this line intersects the polygon boundary at exactly two points, A and B. The point x belongs to the closed segment $\overline{AB}$ (It is obvious that x lies inside the polygon). See figure 4.3.

Let the sum of the weights of the vertices included in one of the closed half-planes, defined by the line through x, say the one indicated in figure 4.3, be less than $\lceil \frac{W}{3} \rceil$. In the chain from B to A, let M be the first vertex (possibly B) and let N be the last vertex (possibly A). The sum of the weights of the vertices included in the chain from M to N is less than $\lceil \frac{W}{3} \rceil$. So $CHAIN(M)$ includes at least the vertex next to N. So $CP(M)$ completely excludes the closed segment $\overline{AB}$ and hence the point x. This contradicts the fact that x belongs to $\mathcal{R}$. □

Theorem 4.2 *$\mathcal{R}$ is the weighted center and hence is nonempty.*

Proof : Follows from lemmas 4.1 and 4.2 and the fact that a weighted centerpoint always exists for any configuration of points with arbitrary weight assignments. □

Though $\mathcal{R}$ is the intersection of n closed half-planes $CP(v_i)$, it may not be necessary to compute all of them. This is because if $CHAIN(v_i)=v_iv_{i+1}\ldots v_j$ and w_j is so large that $CHAIN(v_{i+1})=v_{i+1}v_{i+2}\ldots v_j$, then $CP(v_{i+1})$ will not contribute anything to the intersection

$\mathcal{R}$ and hence need not be generated. We outline a procedure Generate_Closed_Halfplanes() below which generates all the necessary ones in linear time.

Then a point belonging to their intersection which is a weighted centerpoint can be computed in $O(n)$ time using Megiddo's Linear Programming technique [Meg83b]. Further, the whole intersection, and thus the weighted center, can be computed in $O(n \log n)$ time [PS85].

PROCEDURE Generate_Closed_Halfplanes()
Comments: *start* and *end* are two variables. The subroutine *Increment(x)* will make x point to the next vertex in the counterclockwise order.
Begin
1) Initialise *start* and *end* as pointing to v_1.
2) While *true* do
   a) While the sum of the weights of the vertices included between *start* and *end*, both inclusive, is less than $\lceil \frac{W}{3} \rceil$, *Increment*(*end*).
   b) *'Output' CP*(*start*).
   c) *Increment*(*start*) until the sum of the weights of the vertices included between *start* and *end*, both inclusive, becomes just less than $\lceil \frac{W}{3} \rceil$. If *start* becomes v_1 again at any time during the above step, then break.
End.

The above procedure takes $O(n)$ time :

- Step 2*a* executed at most n-1+n-2 = 2n-3 times. (When *start* is at v_n, *end* may be at most at v_{n-2} having completed one full round)
- Step 2*b* executed at most n times.
- Step 2*c* executed exactly n times.

4.4 An algorithm for the general case

In this section, we present an $O(n \log^5 n)$ algorithm for computing a weighted centerpoint of any weighted configuration of points. The algorithm presented is a generalisation of the

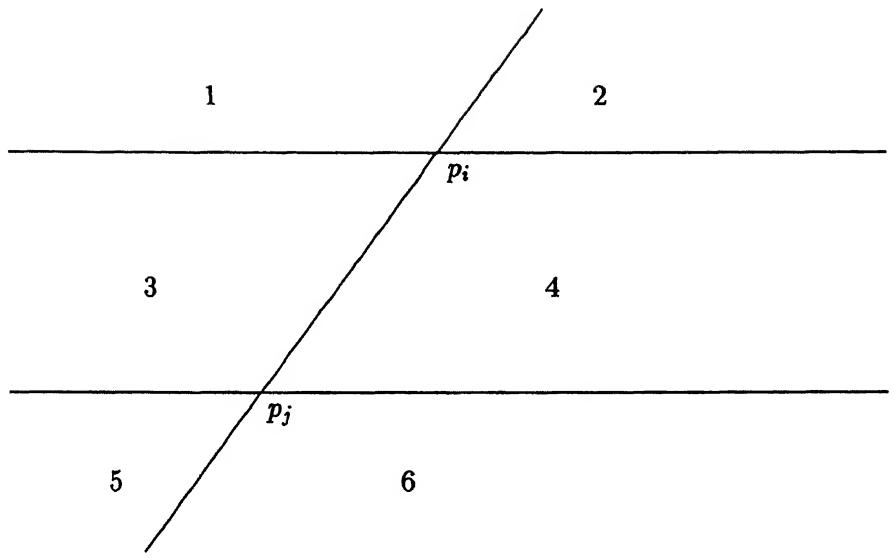


Figure 4.4: The six regions defined by a pair of points

one presented in [CSY87] for computing an ordinary centerpoint. Using Cole's technique [Col87], it can be improved to run in $O(n \log^3 n)$ time.

We are given a set P of n points. Each point p_i has a weight w_i associated with it. Let W be the total weight of all the points. The rotational order of the points in P about a given point p is defined as follows: Consider a horizontal line L through p. A new point set is obtained from P by retaining the points which are on or above L and taking the reflection through p of the points below L. The angular order of this new set about p defines the rotational order of the original set about p.

We now sort the points in P using the rotational order about some point in the weighted center $p^\star$ - which is not known at present but will be determined in the process. We use an m processor, h time parallel algorithm. The relative rotational order of p_i and p_j about the point $p^\star$ depends on which of the six regions $p^\star$ lies in (See figure 4.4). These regions are defined by the horizontal lines through p_i and p_j and the line L_{ij} passing through p_i and p_j.

We use a subroutine, which given an arbitrary line L, tells whether the weighted center intersects L and if so, gives a point on L in the weighted center, and if not determines on which side of L, the weighted center lies. This subroutine takes $O(n \log^3 n)$ time. Using this subroutine, we perform a binary search on the n horizontal lines through the n points in P and determine a pair of consecutive lines between which the weighted center and hence

$p^\star$ lies or find a weighted centerpoint on one of these n lines. This takes $O(n \log^4 n)$ time. Now that we know the two consecutive lines between which the weighted center lies, all that we have to do to resolve the comparison between p_i and p_j is to determine on which side of L_{ij} $p^\star$ lies. We do this by using Megiddo's technique for Linear Programming in fixed dimensions [Meg83b].

In each stage of the sorting algorithm, there are m lines generated, for each of which we have to determine on which side $p^\star$ lies. In $O(n \log^3 n)$ time, we either determine for at least one eighth of the lines on which side $p^\star$ lies or we find a weighted centerpoint in which case we stop. In the former case, we will have found two lines bounding the region in which $p^\star$ lies. By iterating $O(\log m)$ times, we either determine for every line on which side $p^\star$ lies or we find a weighted centerpoint. Also in the former case, we will have determined a region containing $p^\star$ bounded by $O(\log m)$ lines.

To eliminate one eighth of the lines, we proceed as follows: Among the slopes of the m lines, we find the median value. Let it be s. We then pair lines so that each pair contains a line of slope less than s and a line of slope greater than s. For each pair, we compute the intersection point of the lines in the pair, and from among the x-coordinates of the intersection points, we determine the median value x_{med}. We then test whether the line $L : x = x_{med}$ intersects the weighted center and if not, find on which side of L the weighted center lies. If L intersects the weighted center, then we get a point in the weighted center on L and we stop. So without loss of generality, assume that the weighted center lies to the left of L. We then find the line L' of slope s which evenly divides the collection of lines of slope s consisting of those lines of slope s in the original set plus the lines of slope s through the intersection points to the right of L. We determine whether L' intersects the weighted center and if not, find on which side of L' the weighted center lies. Thus we either find a weighted centerpoint or the quadrant in which the weighted center and hence $p^\star$ lies. For every intersection point in the opposite quadrant, one of the two lines through the point will avoid the quadrant in which $p^\star$ lies. These lines can be eliminated from further consideration and they are at least one eighth of the original set.

Thus when the algorithm (using either Preparata's algorithm [Pre78] or the AKS network [AKS83]) terminates, we will have found either a weighted centerpoint or a region $\mathcal{R}$, with respect to which the rotational order of the points in P is fixed. The whole algorithm takes $O(n \log^5 n)$ time. The region $\mathcal{R}$ is the intersection of $h{=}O(\log n)$ regions, one for each

step of the sorting algorithm. Each such region is bounded by $O(\log m) = O(\log n)$ lines. Since the rotational order of the points in P is the same about any point in $\mathcal{R}$, either every point in $\mathcal{R}$ is a weighted centerpoint or none is. Since the weighted center is nonempty for any configuration of points with arbitrary weight assignments, one of the points in $\mathcal{R}$ must be a weighted centerpoint and hence every point is.

We now describe an algorithm, which tests whether the weighted center intersects a given line L, giving a point on L if yes, and if no, determining on which side of L the weighted center lies. We can assume L to be the $x - axis$. For each orientation θ, $-\pi < \theta \leq \pi$, and a point x, let $l_\theta(x)$ denote the directed line through x with orientation θ. Let $f_\theta(x)$ be the ratio of the sum of the weights of the points which lie in the closed half-plane to the right of $l_\theta(x)$ to the total weight W. Define $g(x)=min_\theta f_\theta(x)$, $g_1(x)=min_{0<\theta\leq\pi} f_\theta(x)$, and $g_2(x)=min_{-\pi<\theta\leq 0} f_\theta(x)$.

The following observations are easy :

1. Let L be any line (assumed to be the $x - axis$). Then for each θ, the function $f_\theta(x)$ is monotone as x varies along L. It is constant for $\theta = 0$ or π, decreasing from 1 to 0 for $\theta \in (0,\pi)$ (the 'upward' orientations), and increasing from 0 to 1 for $\theta \in (-\pi,0)$ (the 'downward' orientations).

2. $g_1(x)$ is a decreasing function on L and $g_2(x)$ is an increasing function on L. Let $\mathcal{I}$ be the interval of L over which $g_1(x)=g_2(x)$. This interval will always be non-empty. At all $x \in \mathcal{I}$, $g(x)$ has the same value and g attains its maximum there.

3. For any x, $g(x)$ attains its minimum at some set $\Theta(x)$ of orientations. If $\Theta(x)$ consists only of 'upward' (respectively 'downward') orientations then x lies to the right (resp. left) of $\mathcal{I}$, and conversely. Otherwise x lies in $\mathcal{I}$ and $\Theta(x)$ has both types of orientations.

We seek a point p in $\mathcal{I}$. We search for this by finding the rotational order of the points in P about p, using an m processor, h time parallel algorithm. We compare two points p_i and p_j as follows: Let L_{ij} be the line through them. Let it intersect L in a point, x say. The result of the comparison depends solely on which side of x the point p lies. We rotate a line about x and compute $f_\theta(x)$ as θ varies. Then using the third observation, we determine whether $\mathcal{I}$ includes x or if $\mathcal{I}$ is to the left or to the right of x. This takes $O(n \log n)$. If $\mathcal{I}$ includes x, we choose x itself as p. So either we will resolve the comparison

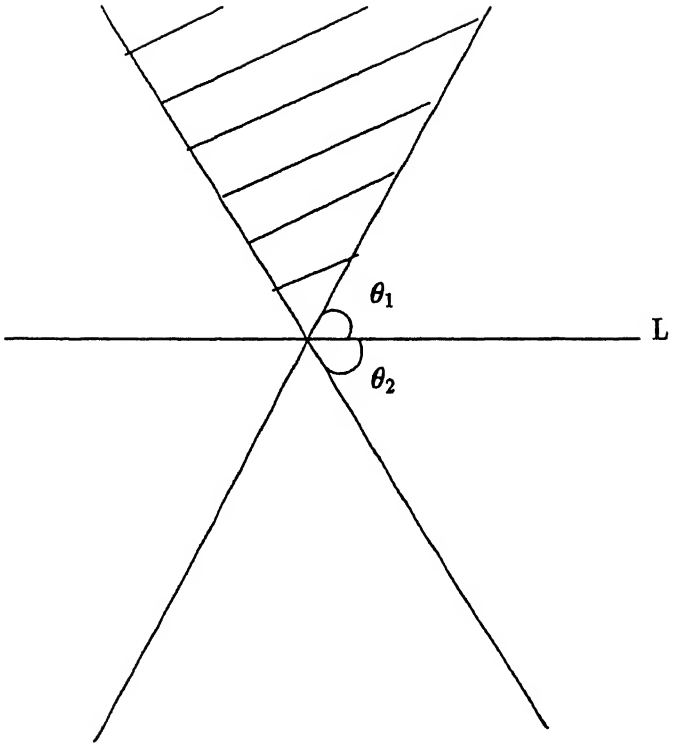


Figure 4.5: Determining the side of L on which the weighted center lies

or find the required point p. To batch m of these comparisons, we sort the resulting m points on L and perform a binary search among them (actually, find the medians etc.). This yields an interval $\mathcal{J}$ (containing p) over which the comparisons performed so far have a fixed result. This takes time $O(n \log n \log m + m)$. Hence the whole algorithm takes $O(h(n \log n \log m + m))$ time. Using either Preparata's algorithm($m = n \log n, h = \log n$) [Pre78] or the AKS network($m = n, h = \log n$) [AKS83], this becomes a time of $O(n \log^3 n)$.

When the algorithm terminates, we either have a point $p \in \mathcal{I}$ or an open interval $\mathcal{J} = (\mathcal{AB})$ containing p with respect to which the rotational order of the points in P is fixed. Neither A nor B can be in $\mathcal{I}$ because in such a case, the algorithm would have output that point. Since the rotational order of P about any point in $\mathcal{J}$ is same, g has the same value throughout $\mathcal{J}$. Since $\mathcal{J}$ contains p, a point of $\mathcal{I}$, it follows that $\mathcal{J} = \mathcal{I}$ and any point in $\mathcal{J}$ will serve as p. So in all cases the algorithm finds p. If $g(p)$ is greater or equal to one third, then p is a weighted centerpoint. Otherwise, p is not a weighted centerpoint and L does not intersect the weighted center. Since $g(p)$ is less than one third, there exist two orientations, an 'upward' θ_1 and a 'downward' θ_2, such that $f_{\theta_1}(p) = f_{\theta_2}(p) < \frac{1}{3}$. See figure 4.5.

It now follows that the weighted center of P lies in the shaded area of the figure 4.5 (the side of L that this shaded area lies in depends on θ_i), because for any point not in this shaded region either the line with orientation θ_1 or the one with orientation θ_2 is guaranteed to have points whose total weight is less than $\lceil \frac{W}{3} \rceil$ to its right. Thus if L does not intersect the weighted center, we can determine on which side of L, the weighted center lies.

Chapter 5

Conclusion

This thesis is on the centerpoint problem in Computational Geometry. Efficient algorithms for computing a centerpoint in two or higher dimensions have important applications. The following issues were discussed in this thesis :

1. The application of computing a centerpoint in d-dimensions for finding graph separators was discussed in chapter 2. A new class of graphs, known as overlap graphs, which includes many other classes of graphs which find applications in Numerical Analysis and Computational Geometry as special cases, was introduced. Next a randomised algorithm was presented to compute a small graph separator for any overlap graph. The use of finding graph separators in Numerical Analysis was thereafter outlined - for solving sparse linear systems of equations.

2. A cost optimal parallel algorithm in the exclusive read exclusive write parallel random access machine model, for finding a centerpoint in two dimensions was presented in chapter 3. The time complexity was $O(\log^3 n \log^* n)$ and the number of processors required was $\frac{n}{\log^3 n \log^* n}$. In addition, a cost optimal parallel algorithm for finding a generalised ham-sandwich cut in two dimensions (linearly separated case) was presented, which was used as a subroutine in the centerpoint algorithm. The technique used is quite general and can be used to parallelise other prune-and-search algorithms as well. For example, we can give an optimal parallel algorithm in the EREW model, taking $O(\log^2 n \log^* n)$ time for the two dimensional linear programming problem. However, X. Deng [Den90] has given an optimal $O(\log n)$ time parallel algorithm for the above problem in the CRCW model. He observed that we do not need the full power of the

median for discarding a constant proportion of the constraints [Meg83b, Dy84] and used Cole's approximate median finding algorithm [Col88]. Whether incorporating approximation ideas (we need to define an approximate ham-sandwich cut) in the centerpoint algorithm leads to similar improvements is not known.

3. In chapter 4, the notion of a centerpoint was extended to that of a weighted centerpoint. It was shown that a weighted centerpoint always exists for any configuration of points with arbitrary weight assignments. A linear time algorithm for finding a weighted centerpoint of a configuration of points that form the vertices of a convex polygon was also proposed. Next, an $O(n \log^3 n)$ algorithm for computing a weighted centerpoint of any weighted configuration of points was given; whether there is a linear time algorithm for this case is not known.

Bibliography

[AG73] A. George : Nested dissection of a regular finite element mesh, Siam Journal on Numerical Analysis, Vol.10, No.2, April 1973, 345-363.

[AHU74] A.V.Aho, J.Hopcroft and J.D.Ullman : *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[AKS83] M.Ajtai, J.Komlós and E.Szemerédi : Sorting in $c \log n$ parallel steps, Combinatorica, Vol.3, No.1, 1983, 1-19.

[Br74] R.P.Brent : The parallel evaluation of general arithmetic expressions, Journal of the Association for Computing Machinery, Vol.21, No.2, April 1974, 201-206.

[Col87] R.Cole : Slowing down sorting networks to obtain faster sorting algorithms, Journal of the Association for Computing Machinery, Vol.34, No.1, January 1987, 200-208.

[Col88] R.Cole : An optimally efficient selection algorithm, Information Processing Letters, Vol.26, No.6, January 1988, 295-299.

[CSY87] R.Cole, M.Sharir and C.K.Yap : On k-hulls and related problems, Siam Journal on Computing, Vol.16, No.1, February 1987, 61-77.

[CY85] R. Cole and C.K.Yap : A parallel median algorithm, Information Processing Letters, Vol.20, No.3, April 1985, 137-139.

[Den90] X. Deng : An optimal parallel algorithm for linear programming in the plane, Information Processing Letters, Vol.35, No.4, August 1990, 213-217.

[Dy84] M.E.Dyer : Linear time algorithms for two and three variable linear programs, Siam Journal on Computing, Vol.13, No.1, February 1984, 31-45.

[Ede87] H.Edelsbrunner : *Algorithms in Combinatorial Geometry*, Springer-Verlag, 1987.

[JM93] S.Jadhav and A.Mukhopadhyay : Computing a centerpoint of a finite planar set of points in linear time, 9^{th} ACM Symposium on Computational Geometry, 1993, 83-90.

[LR79] R.J.Lipton and R.E.Tarjan : A separator theorem for planar graphs, Siam Journal on Applied Mathematics, Vol.36, No.2, April 1979, 177-189.

[LRT79] R.J.Lipton, D.J.Rose and R.E.Tarjan : Generalised nested dissection, Siam Journal on Numerical Analysis, Vol.16, No.2, April 1979, 346-358.

[LS90] Chi-Yuan Lo and William Steiger : An optimal time algorithm for ham-sandwich cuts in the plane, 2^{nd} Canadian Conference on Computational Geometry, 1990, 5-9.

[Mat91] J. Matoušek : Approximations and optimal geometric divide-and-conquer, 23^{rd} ACM Symposium on Theory of Computing, 1991, 506-511.

[Mat92] J. Matoušek : Computing the center of planar point sets, Discrete and Computational Geometry: Papers from the dimacs special year, American Mathematical Society, J.E.Goodman, R.Pollack, W.Steiger, Eds, 1992, 221-230.

[Meg83a] N.Megiddo : Applying parallel computation algorithms in the design of serial algorithms, Journal of the Association for Computing Machinery, Vol.30, No.4, October 1983, 852-865.

[Meg83b] N.Megiddo : Linear time algorithms for linear programming in R^3 and related problems, Siam Journal on Computing, Vol.12, No.4, November 1983, 759-776.

[Meg85] N.Megiddo : Partitioning with two lines in the plane, Journal of Algorithms, Vol.6, No.3, September 1985, 430-433.

[MT90] G.L.Miller and W.Thurston : Separators in two and three dimensions, 22^{nd} ACM Symposium on Theory of Computing, 1990, 300-309.

[NS90] N.Naor and M.Sharir : Computing a point in the center of a point set in three dimensions, 2^{nd} Canadian Conference on Computational Geometry, 1990, 10-13.

[Pre78] F.Preparata : New parallel sorting schemes, IEEE Transactions on Computers, Vol.C-27, No.7, July 1978, 669-673.

[PS85] F.P.Preparata and M.I.Shamos : *Computational Geometry- An Introduction*, Springer-Verlag, 1985.

[Rei78] Angelika Reiser : A linear selection algorithm for sets of elements with weights, Information Processing Letters, Vol.7, No.3, April 1978, 159-162.

[Ros73] D.J.Rose : A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations, *Graph Theory and Computing*, R.Read, ed., Academic Press, New York, 1973, 183-217.

[RTL76] D.J.Rose, R.E.Tarjan and G.S.Leuker : Algorithmic aspects of vertex elimination on graphs, Siam Journal on Computing, Vol.5, No.2, June 1976, 266-283.

[Tng91] Shang-Hua Teng : Points, spheres and separators - A unified geometric approach to graph partitioning, Ph.D. thesis, Carnegie Mellon University, August 1991.

[YB61] I.M.Yaglom and V.G.Boltyanskii,*Convex Figures*, (trans.) Holt, Rinehart and Winston, New York, 1961.

Appendix A

The Ham-sandwich Cut Problem in the Dual Plane

In this appendix, we introduce one specific geometric transform that can be used to map a set of points P in the plane to a set of lines L in the plane or *vice versa*, such that certain statements about P are true iff their corresponding dual statements about L are true. We then describe the ham-sandwich cut problem in the dual plane.

A.1 A geometric transform

Let $p = (\pi_1, \pi_2)$ be a point in E^2. Transform $\mathcal{D}$ maps p to the line $\mathcal{D}(p)$ given by the equation $y = 2\pi_1 x - \pi_2$ and *vice versa*, that is, it maps a nonvertical line l to a point $\mathcal{D}(l)$ such that $\mathcal{D}(\mathcal{D}(l)) = l$.

If P is a set of points, then $\mathcal{D}(P) = \{\mathcal{D}(p) \mid p \in P\}$. $\mathcal{D}$ is extended to sets of lines similarly. Note that the x-coordinate of p determines the slope of the line $\mathcal{D}(p)$.

The following observations about $\mathcal{D}$ can be easily verified :

Let p be a point in E^2 and let l be a nonvertical line in E^2.

- *Incidence preservation* : point p belongs to line l iff point $\mathcal{D}(l)$ belongs to line $\mathcal{D}(p)$.
- *Order preservation* : point p lies above(below) line l iff point $\mathcal{D}(l)$ lies above(below) line $\mathcal{D}(p)$.

A.2 The notion of an arrangement of lines and levels in arrangements

A finite set L of lines in E^2 defines a dissection of E^2 into connected components of dimensions two, one and zero. We call this dissection the arrangement $\mathcal{A}(L)$ of L.

Notation: Given a nonvertical line l, l^+ denotes the open half-plane lying above l and l^- denotes the open half-plane lying below l.

Given an arrangement $\mathcal{A}(L)$ of n nonvertical lines in the plane, we define the following integer functions for each point p in the plane :

- $a_L(p)$: $p \in l^-$ for $a_L(p)$ lines l of L.
- $o_L(p)$: p lies on l for $o_L(p)$ lines l of L.
- $b_L(p)$: $p \in l^+$ for $b_L(p)$ lines l of L.

The k-level L_k of the above arrangement is defined as the set of points p such that $a_L(p) < k$ and $a_L(p)+o_L(p) \geq k$, for each $1 \leq k \leq n$. Intuitively, it is a continuous, piecewise linear function whose segments always coincide with one of the lines in the arrangement.

A.3 The ham-sandwich cut problem in the dual plane

We are given two sets of points in the plane : P with m points and Q with n points. These two sets are separated by a line with no point lying on the separating line. We are interested in finding a nonvertical line l such that the number of points of P (Q) contained in l^- is less than p (q), but when the points lying on l are counted, it is greater or equal to p (q).

We dualise the problem. Without loss of generality, we can assume that the separating line is the y-axis in the primal plane with P lying to its left and Q to its right. Then in the dual plane, all lines of $\mathcal{D}(P)$ will have negative slopes and all lines of $\mathcal{D}(Q)$ will have positive slopes. Any level of $\mathcal{A}(\mathcal{D}(P))$ is a monotonically decreasing function and that of $\mathcal{A}(\mathcal{D}(Q))$ is a monotonically increasing function and hence always intersect.

Let r denote the point where the p^{th} level of $\mathcal{A}(\mathcal{D}(P))$ and the q^{th} level of $\mathcal{A}(\mathcal{D}(Q))$ meet. Since r lies on the p^{th} level of $\mathcal{A}(\mathcal{D}(P))$, the number of lines of $\mathcal{D}(P)$ lying above r is less than p but the number of lines lying above or passing through r is greater than or equal to p. A similar observation holds regarding the disposition of the lines of $\mathcal{D}(Q)$ with

respect to r. These observations imply that in the primal plane, $\mathcal{D}(r)^-$ contains less than p points of P and q points of Q but with the points lying on $\mathcal{D}(r)$ included, it contains at least p points of P and q points of Q, that is, $\mathcal{D}(r)$ is the line we are looking for. Observe that $\mathcal{D}(r)$ contains exactly one point of P and Q each, if $P \cup Q$ is in general position.

Appendix B

Detailed Analyses of the Parallel Algorithms

In this appendix, we give detailed analyses of the parallel algorithms presented in chapter 3. We analyse certain steps in the ham-sandwich cut algorithm in detail - in particular, the subroutine for determining on which side of a given line L, the point I lies (hereafter, referred to as line query) and step 1 of the pruning phase. The notations and the symbols used are the same ones used in section 3.3 of chapter 3.

The first pruning phase of the ham-sandwich cut algorithm, as described, employs $\frac{N}{\log N \log^* N}$ processors. Since the model of computation is EREW, these processors can not simultaneously read a memory location. Hence the values of certain variables such as N, p etc which are needed by all the processors should be communicated to them and the time required for this should be taken into account.

The line query routine is implemented as follows :

1. The values of N, m, n, p, q, α and θ are communicated to all the processors. This takes $O(\log(number\_of\_processors)) < O(\log N)$ time.

2. To compute the intersections of the m lines in B with L, these m lines are divided into $\frac{m}{\log N \log^* N}$ groups of size $\log N \log^* N$ each and each group is assigned to one processor ($\frac{N-m}{\log N \log^* N}$ processors remain idle) which computes the intersections of the lines in that group with L. This takes $\log N \log^* N$ time.

3. For selecting x', we use Cole's selection algorithm [Col88] which can be implemented

in $O(\log m \log^* m)$ time using $\frac{m}{\log m \log^* m}$ processors. If the number of available processors is less than this, that is, $\frac{N}{\log N \log^* N} < \frac{m}{\log m \log^* m}$, then the selection algorithm is rescheduled using $\frac{N}{\log N \log^* N}$ processors which takes $O(\frac{m}{\log m \log^* m} \times \frac{\log N \log^* N}{N} \times \log m \log^* m)$ time i.e. $O(\log N \log^* N)$ time. This x' needs to be communicated to all the processors which takes less than $\log N$ time.

4. The remaining steps can similarly be implemented in $O(\log N \log^* N)$ time.

It is clear that the line query routine can be implemented in $O(\log N \log^* N)$ time using $\frac{N}{\log N \log^* N}$ processors.

We now describe how to implement step 1 of the pruning phase in detail :

1. The sets A and B are merged into a temporary array T of size N. Set A is divided into $\frac{n}{\log N \log^* N}$ groups of size $\log N \log^* N$ each; each group is assigned to one processor which copies the lines in that group into T. B is similarly copied into T. This takes $2 \log N \log^* N$ time.

2. s is found using Cole's selection algorithm which takes $O(\log N \log^* N)$ time.

3. Now two sets should be formed, one with lines of slope greater than s and the other with lines of slope smaller than s. The first set is formed as follows :

   (a) T is divided into $\frac{N}{\log N \log^* N}$ groups of size $\log N \log^* N$ each and each group is assigned to a processor. The i^{th} processor counts the number of lines in group i with slope greater than s and writes that count in location i in an array, say *Count*. This takes $\log N \log^* N$ time.

   (b) The prefix sum of the entries in *Count* is computed. The result is stored back in *Count*. *Count* has $\frac{N}{\log N \log^* N}$ entries and the prefix sum of these can be computed in $\log(\frac{N}{\log N \log^* N}) < \log N$ time using $\frac{N}{\log N \log^* N}$ processors.

   (c) Now processor i scans its group again and selects lines with slope greater than s and moves them into an array (meant for storing these lines), using the entry in location $i-1$ of *Count* as index.

The second set is formed similarly. The time taken by these operations is $O(\log N \log^* N)$.

4. Once the two sets are formed, lines - one from each set - can be paired and their point of intersection computed in $\log N \log^* N$ time.

Hence the total time taken by step 1 of the pruning phase is $O(\log N \log^* N)$.

We have described the ham-sandwich cut algorithm as if different number of processors are available for the different stages of the algorithm. In the implementation described, stage i of the algorithm would use $\frac{\beta^{i-1}N}{\log(\beta^{i-1}N)\log^*(\beta^{i-1}N)}$ processors and take $\log(\beta^{i-1}N)\log^*(\beta^{i-1}N)$ time. To implement the algorithm using $\frac{N}{\log^2 N \log^* N}$ processors, we have to reschedule certain stages of the algorithm. Stage i is implemented as follows : The control processor compares $a = \frac{N}{\log^2 N \log^* N}$ with $b_i = \frac{\beta^{i-1}N}{\log(\beta^{i-1}N)\log^*(\beta^{i-1}N)}$. If $a \geq b_i$, then stage i is not rescheduled - so it runs in $\log(\beta^{i-1}N)\log^*(\beta^{i-1}N)$ time using b_i processors. If $a < b_i$, then stage i is rescheduled using a processors. It is easy to verify that the time taken by the rescheduled stage i is $O(\beta^{i-1}\log^2 N \log^* N + \log(number\_of\_processors))$. Since the $number\_of\_processors = a$ is less than b_i, we have $\log(a) < \log(b_i) < \log(\beta^{i-1}N) < \log(\beta^{i-1}N)\log^*(\beta^{i-1}N)$. In either case, the time required by stage i is $O(\beta^{i-1}\log^2 N \log^* N + \log(\beta^{i-1}N)\log^*(\beta^{i-1}N))$. It follows that the ham-sandwich cut algorithm can be implemented in $O(\log^2 N \log^* N)$ time.

Appendix C

The Pruning Step of the Centerpoint Algorithm in More Detail

In this appendix, we explain the pruning step of the parallel centerpoint algorithm presented in chapter 3 in more detail. The symbols and notations used in this appendix are the same ones introduced in section 3.4 of chapter 3.

We denote the four sets involved in pruning S_{LU}, S_{LD}, S_{RD} and S_{RU} by $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ respectively. Sets $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{D}$ contain $\lfloor \frac{n}{12} \rfloor - 1$ points each. We assume that $\mathcal{C}$ contains exactly the same number of points. We make two observations below. We will not give formal proofs of these observations but only verify them by considering the various possible arrangements of the four planes L, U, D and R.

Observation 1 : Sets $\mathcal{A}$ and $\mathcal{C}$ do not have any common point of P. Similarly sets $\mathcal{B}$ and $\mathcal{D}$ are disjoint. We verify this observation by noting that no point of P belongs to all the four planes L, U, D and R in any arrangement of them.
Observation 2 : There is a set disjoint from the other three sets.

The pruning step in the parallel algorithm of chapter 3 is implemented as follows :

1. Find four bit strings s_A, s_B, s_C and s_D corresponding to sets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ respectively. In each bit string, bit i is set if point i belongs to that set.

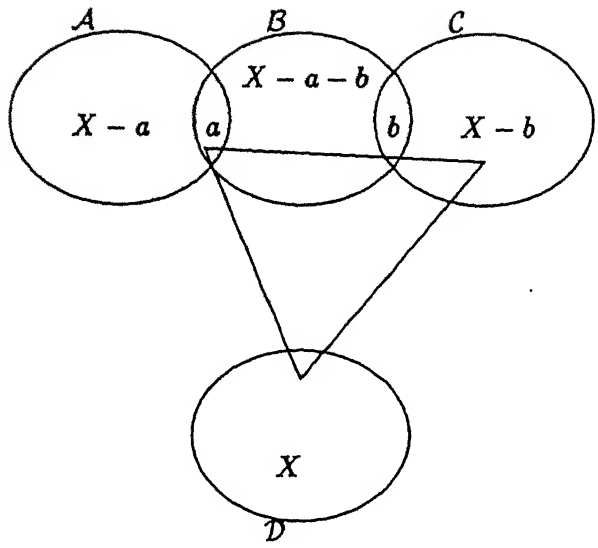


Figure C.1: Eliminating triplets of points in *case 2*

2. Find the following four intersections of these bit strings - s_{AB} (intersection of s_A and s_B), s_{BC}, s_{CD} and s_{DA}. It follows from the above two observations that following three cases can arise :

   - *Case 2:* Two of these intersections are non-null.
   - *Case 1:* One of these intersections is non-null.
   - *Case 0:* All the four sets are disjoint.

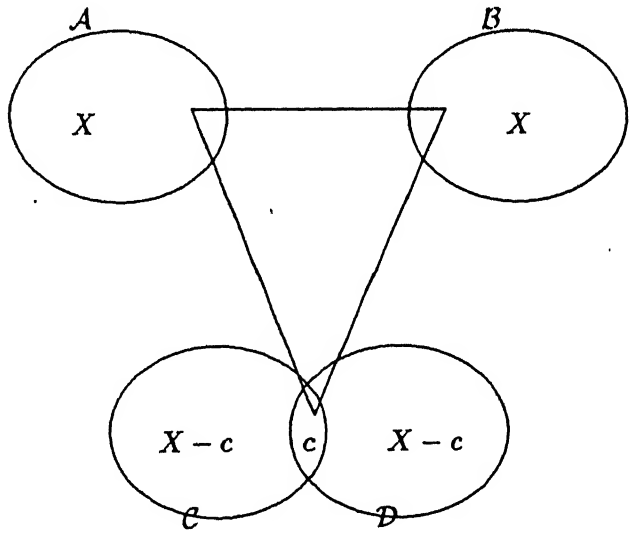


Figure C.2: Eliminating triplets of points in *case 1*

3. *Case 2:* Let $a = |\mathcal{A} \cap \mathcal{B}|$, $b = |\mathcal{B} \cap \mathcal{C}|$ and $X = |\mathcal{D}|$. Observe that $X - a \geq b$ (which

implies that $X - b \geq a$). Let $a \leq b$. Eliminate triplets of points as shown in figure C.1 until all a points in $\mathcal{A} \cap \mathcal{B}$ get exhausted. This now reduces to *case 1*.

Case 1: Let $c = |\mathcal{C} \cap \mathcal{D}|$ and $X = |\mathcal{A}|$. Eliminate triplets of points as shown in figure C.2 until $\mathcal{C} \cap \mathcal{D}$ gets exhausted. This now reduces to *case 0*.

Case 0: Four points, one from each of the sets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$, are chosen and joined in that order. If they form a convex quadrilateral, they are replaced by the point of intersection of the diagonals of the quadrilateral. If they form a non-convex quadrilateral, the interior point is retained and other three are discarded.

It is clear that the above pruning routine can be implemented using $\frac{n}{\log^2 n \log^* n}$ processors in time $O(\log^2 n \log^* n)$. It also ensures that a fixed fraction of the points of P is discarded in each pruning phase.

We now verify the two observations made before by considering the various possible arrangements of the four half-planes L, U, D and R. We denote the lines defining these half-planes by l, u, d and r respectively. Without loss of generality, we can assume that the line l is vertical. Lines u and d will be nonvertical. The point of intersection of l and u is denoted by p_{lu}. It is clear that the half-plane D can not contain the point p_{lu} i.e. d intersects l below the point p_{lu}. Similarly r intersects u at a point p_{ru} lying to the right of l.

Three cases arise depending on the slopes of u and d :

- Case 1 : u and d are parallel.
- Case 2 : u and d intersect to the left of l.
- Case 3 : u and d intersect to the right of l.

In each of the above cases, we consider the different positions which r can take and verify the observations in each of the resulting arrangements. Let α, $0 < \alpha < \pi$, denote the angle between u and r, measured as r rotates about the point p_{ru} in the counterclockwise direction.

Figure C.3 shows a few of the possible arrangements in case 1 as α takes different values. Since $U \cap D = \phi$, observation 1 is satisfied. In the first three arrangements, $\mathcal{A} = S_{LU}$ is disjoint from the remaining three sets and in the last one, $\mathcal{B}$ is disjoint from the remaining three sets.

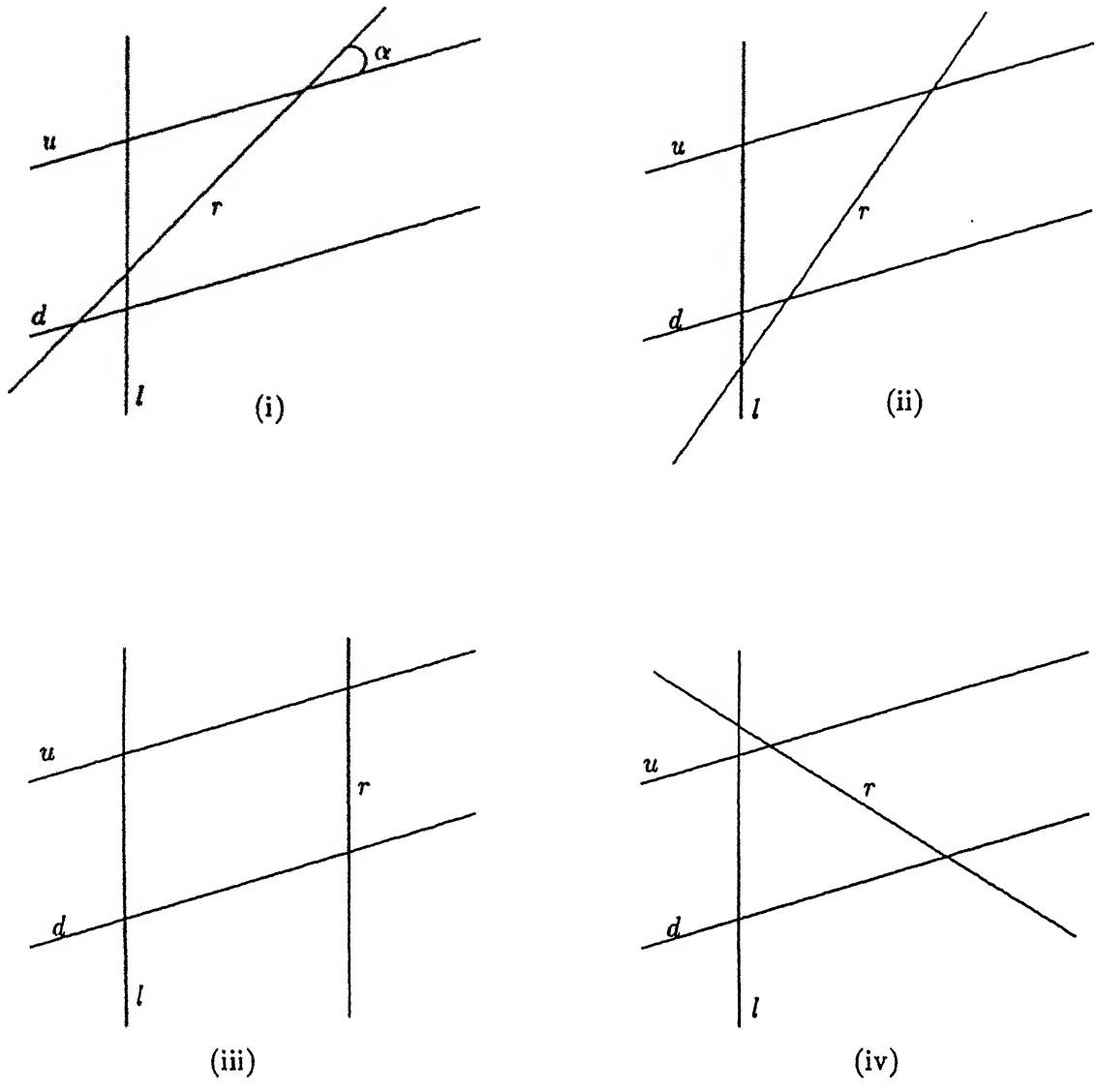
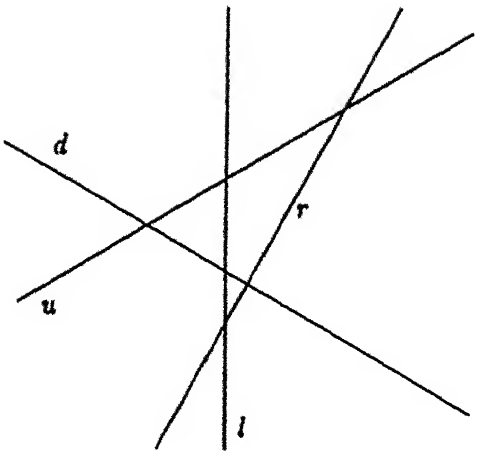
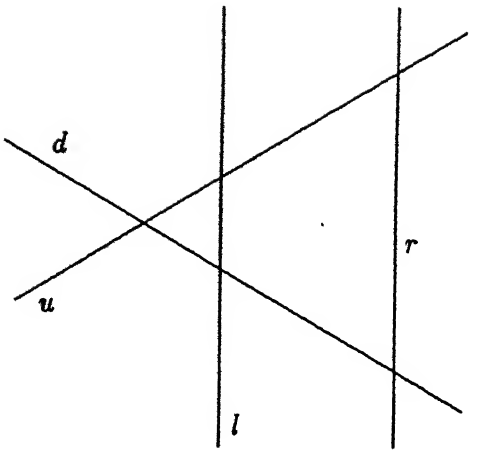


Figure C.3: Different arrangements when u and d are parallel

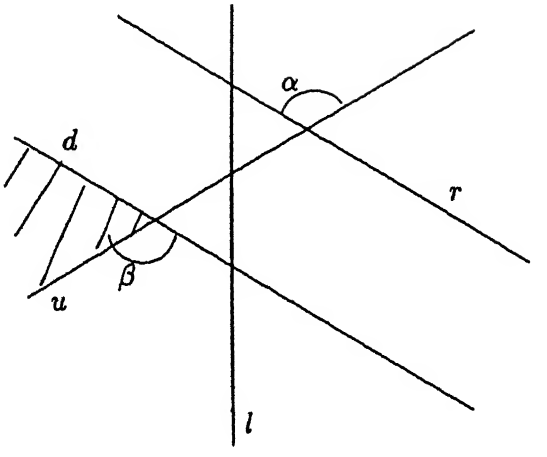
Figure C.4 some of the possible arrangements in case 2. In this case, $0 < \alpha < \beta$ where β is the angle shown in figure C.4(iii). Since $\alpha < \beta$ (because $(P - L) \cap (P - U) \cap (P - D)$ contains $\frac{n}{6}$ points and R is required to contain $\frac{n}{4}$ points of $P - U$), R does not contain any point of $L \cap U \cap D$ (shown shaded in figure C.4(iii)), verifying observation 1. Observation 2 can be easily verified.



(i)



(ii)



(iii)

Figure C.4: Different arrangements when u and d intersect to the left of l

In case 3 where u and d intersect to the right of l, $L \cap U \cap D = \phi$ and hence observation 1 is true. The second observation can be verified by considering the different orientations possible for r.